



**Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)**

**Факультет «Информатика и системы управления»
Кафедра «Системы обработки информации и управления»
Курс «Базовые компоненты интернет-технологий»**

**Отчёт по лабораторной работе №3-4
«Функциональные возможности языка Python.»**

**Выполнил:
студент группы ИУ5-33Б
Рыбин Владислав**

**Проверил:
к.т.н., доц., Ю. Е. Гапанюк**

2022 г.

Задание:

Задание лабораторной работы состоит из решения нескольких задач.

Файлы, содержащие решения отдельных задач, должны располагаться в пакете `lab_python_fr`. Решение каждой задачи должно располагаться в отдельном файле.

При запуске каждого файла выдаются тестовые результаты выполнения соответствующего задания.

Задача 1 (файл `field.py`)

Необходимо реализовать генератор `field`. Генератор `field` последовательно выдает значения ключей словаря.

В качестве первого аргумента генератор принимает список словарей, дальше через `*args` генератор принимает неограниченное количество аргументов.

- Если передан один аргумент, генератор последовательно выдает только значения полей, если значение поля равно `None`, то элемент пропускается.
- Если передано несколько аргументов, то последовательно выдаются словари, содержащие данные элементы. Если поле равно `None`, то оно пропускается. Если все поля содержат значения `None`, то пропускается элемент целиком.

Текст программы

```
def field(items, *args):
    assert len(args) > 0, "Количество искомых аргументов равно 0"

    lst = []
    if len(args) > 1:
        for item in items:
            yield {arg: item[arg] for arg in args}
    else:
        for item in items:
            yield item[args[0]]

    return lst

def main():
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}
    ]

    # должен выдавать 'Ковер', 'Диван для отдыха'
    a = []
    for i in field(goods, 'title'):
```

```

        a.append(i)
    print(a)

    # должен выдавать {'title': 'Ковер', 'price': 2000}, {'title': 'Диван для
отдыха', 'price': 5300}
    a = []
    for i in field(goods, 'title', 'price'):
        a.append(i)
    print(a)

if __name__ == "__main__":
    main()

```

Пример выполнения

```

['Ковер', 'Диван для отдыха']
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}]

Process finished with exit code 0

```

Задача 2 (файл *gen_random.py*)

Необходимо реализовать генератор `gen_random(количество, минимум, максимум)`, который последовательно выдает заданное количество случайных чисел в заданном диапазоне от минимума до максимума, включая границы диапазона.

Текст программы

```

from random import randint

def gen_random(num_count, begin, end):
    for num in range(num_count):
        yield randint(begin, end)

if __name__ == '__main__':
    print(list(gen_random(5, 1, 3)))

```

Пример выполнения

```
[1, 2, 2, 1, 1]
```

Задача 3 (файл *unique.py*)

- Необходимо реализовать итератор `Unique(данные)`, который принимает на вход массив или генератор и итерируется по элементам, пропуская дубликаты.
- Конструктор итератора также принимает на вход именованный `bool`-параметр `ignore_case`, в зависимости от значения которого будут

считаться одинаковыми строки в разном регистре. По умолчанию этот параметр равен False.

- При реализации необходимо использовать конструкцию ****kwargs**.
- Итератор должен поддерживать работу как со списками, так и с генераторами.
- Итератор не должен модифицировать возвращаемые значения.

Текст программы

```
from lab_python_fp.gen_random import gen_random

class Unique(object):
    def __init__(self, items, **kwargs):
        self.ignore_case = kwargs.get('ignore_case', False)
        self.set = set()
        self.iter = iter(items)

    def __next__(self):
        for current in self.iter:
            if self.ignore_case == False:
                if current not in self.set:
                    self.set.add(current)
                    return current
            else:
                if current.lower() not in self.set:
                    self.set.add(current)
                    return current
        raise StopIteration

    def __iter__(self):
        return self

class Unique(object):
    def __init__(self, items, **kwargs):
        self.ignore_case = kwargs.get("ignore_case", False)
        self.iter = iter(items)
        self.used = set()

    def __next__(self):
        for current in self.iter:
            if isinstance(current, str):
                if (current.lower() if self.ignore_case else current) not in self.used:
                    self.used.add(current.lower())
                    return current
            else:
                if current not in self.used:
                    self.used.add(current)
                    return current
        raise StopIteration

    def __iter__(self):
        return self

def main():

    u = Unique([2, 1, 2, 3, 3, 3, 3, 2, 2, 2, 2, 1, 1, 4, 5, 4])
```

```

for i in u:
    print(i, end=" ")
print()
u = Unique(["A", "a", "B", "b", "b", "B", "A", "C"])
for i in u:
    print(i, end=" ")
print()
u = Unique(["A", "a", "B", "b", "b", "B", "A", "C"], ignore_case=True)
for i in u:
    print(i, end=" ")
print()

if __name__ == "__main__":
    print(list(Unique([1, 1, 1, 1, 1, 2, 2, 2, 2, 2])))
    print(list(Unique(gen_random(10, 1, 3))))
    print(list(Unique(['a', 'A', 'b', 'B', 'a', 'b'])))
    print(list(Unique(['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B'],
ignore case=True)))

```

Пример выполнения

```

[1, 2]
[2, 3, 1]
['a', 'A', 'b', 'B']
['a', 'b']

Process finished with exit code 0

```

Задача 4 (файл sort.py)

Дан массив 1, содержащий положительные и отрицательные числа. Необходимо **одной строкой кода** вывести на экран массив 2, которые содержит значения массива 1, отсортированные по модулю в порядке убывания. Сортировку необходимо осуществлять с помощью функции sorted.

Необходимо решить задачу двумя способами:

1. С использованием lambda-функции.
2. Без использования lambda-функции.

Текст программы

```

data = [4, -30, 100, -100, 123, 1, 0, -1, -4]

if __name__ == '__main__':
    result = sorted(data, key=abs, reverse=True)
    print(result)

    result_with_lambda = sorted(data, key=lambda a: a if a > 0 else -a,
reverse=True)
    print(result_with_lambda)

```

Пример выполнения

```
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]

Process finished with exit code 0
```

Задача 5 (файл `print_result.py`)

Необходимо реализовать декоратор `print_result`, который выводит на экран результат выполнения функции.

- Декоратор должен принимать на вход функцию, вызывать её, печатать в консоль имя функции и результат выполнения, после чего возвращать результат выполнения.
- Если функция вернула список (`list`), то значения элементов списка должны выводиться в столбик.
- Если функция вернула словарь (`dict`), то ключи и значения должны выводиться в столбик через знак равенства.

Текст программы

```
def print_result(func):
    def read_func(*args, **kwargs):
        print(func.__name__)
        result = func(*args, **kwargs)
        if type(result) is list:
            for item in result:
                print(item)
        elif type(result) is dict:
            for key, item in result.items():
                print(f'{key} = {item}')
        else:
            print(result)
        return result
    return read_func

@print_result
def test_1():
    return 1

@print_result
def test_2():
    return 'iu5'

@print_result
def test_3():
    return {'a': 1, 'b': 2}
```

```

@print_result
def test_4():
    return [1, 2]

if __name__ == '__main__':
    print('!!!!!!!')
    test_1()
    test_2()
    test_3()
    test_4()

```

Пример выполнения

```

!!!!!!!
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2

Process finished with exit code 0

```

Задача 6 (файл cm_timer.py)

Необходимо написать контекстные менеджеры `cm_timer_1` и `cm_timer_2`, которые считают время работы блока кода и выводят его на экран.

`cm_timer_1` и `cm_timer_2` реализуют одинаковую функциональность, но должны быть реализованы двумя различными способами (на основе класса и с использованием библиотеки `contextlib`).

Текст программы

```

from time import time, sleep
from contextlib import contextmanager

class cm_timer_1:
    def __init__(self):
        self.start = 0
        self.stop = 0

    def __enter__(self):

```

```

        self.start = time()

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.stop = time()
        print(f'Time: {self.stop - self.start}')

@contextmanager
def cm_timer_2():
    start_time = time()
    yield None
    end_time = time()
    print(f'Time: {end_time - start_time}')

if __name__ == '__main__':
    with cm_timer_1():
        sleep(2.5)

    with cm_timer_2():
        sleep(2.5)

```

Пример выполнения

```

Time: 2.51674485206604
Time: 2.503153085708618

Process finished with exit code 0

```

Задача 7 (файл *process_data.py*)

- В предыдущих задачах были написаны все требуемые инструменты для работы с данными. Применим их на реальном примере.
- В файле *data_light.json* содержится фрагмент списка вакансий.
- Структура данных представляет собой список словарей с множеством полей: название работы, место, уровень зарплаты и т.д.
- Необходимо реализовать 4 функции - *f1*, *f2*, *f3*, *f4*. Каждая функция вызывается, принимая на вход результат работы предыдущей. За счет декоратора *@print_result* печатается результат, а контекстный менеджер *cm_timer_1* выводит время работы цепочки функций.
- Предполагается, что функции *f1*, *f2*, *f3* будут реализованы в одну строку. В реализации функции *f4* может быть до 3 строк.
- Функция *f1* должна вывести отсортированный список профессий без повторений (строки в разном регистре считать равными). Сортировка должна игнорировать регистр. Используйте наработки из предыдущих задач.

- Функция f2 должна фильтровать входной массив и возвращать только те элементы, которые начинаются со слова “программист”. Для фильтрации используйте функцию filter.
- Функция f3 должна модифицировать каждый элемент массива, добавив строку “с опытом Python” (все программисты должны быть знакомы с Python). Пример: Программист С# с опытом Python. Для модификации используйте функцию map.
- Функция f4 должна сгенерировать для каждой специальности зарплату от 100 000 до 200 000 рублей и присоединить её к названию специальности. Пример: Программист С# с опытом Python, зарплата 137287 руб. Используйте zip для обработки пары специальность — зарплата.

Текст программы

```
import json
from cm_timer import cm_timer_1
from field import field
from gen_random import gen_random
from print_result import print_result
from unique import Unique

path = r'data_light.json'

with open(path, encoding="utf-8") as f:
    data = json.load(f)

@print_result
def f1(arg):
    return sorted(Unique(field(arg, "job-name"), ignore_case=True),
key=lambda s: s.lower())

@print_result
def f2(arg):
    return list(filter(lambda s: s[:len("программист")].lower() ==
"программист", arg))

@print_result
def f3(arg):
    return list(map(lambda s: s + " с опытом Python", arg))

@print_result
def f4(arg):
    z = zip(arg, gen_random(len(arg), 100000, 200000))
    return [f"{profession}, зарплата {salary} руб." for profession, salary in
z]

if __name__ == '__main__':
    with cm_timer_1():
        f4(f3(f2(f1(data))))
```

```
a = [1, 2, 3, 4]
a_squares = [i*i for i in a]
b = list(zip(a, a_squares))
c = [(i, i*i) for i in a]
e = list(map(lambda i: (i, i*i), a))
print(a)
print(b)
print(c)
print(e)
```

Пример выполнения

```
f4
Программист с опытом Python, зарплата 136501 руб.
Программист / Senior Developer с опытом Python, зарплата 199797 руб.
Программист 1C с опытом Python, зарплата 163949 руб.
Программист C# с опытом Python, зарплата 100800 руб.
Программист C++ с опытом Python, зарплата 121273 руб.
Программист C++/C#/Java с опытом Python, зарплата 179579 руб.
Программист/ Junior Developer с опытом Python, зарплата 174290 руб.
Программист/ технический специалист с опытом Python, зарплата 153564 руб.
Программист-разработчик информационных систем с опытом Python, зарплата 117590 руб.
Time: 0.0270082950592041
[1, 2, 3, 4]
[(1, 1), (2, 4), (3, 9), (4, 16)]
[(1, 1), (2, 4), (3, 9), (4, 16)]
[(1, 1), (2, 4), (3, 9), (4, 16)]

Process finished with exit code 0
```